Task-Oriented Design of Virtual Worlds

Chris Stary

Vienna University of Technology
CD-Lab for Expert Systems
Paniglgasse 16, A- 1040 Vienna, Austria
e-mail: stary@vexpert.dbai.tuwien.ac.at

Abstract

In this paper a design methodology for task-oriented virtual world applications is proposed. An object-oriented approach is used for developing an agent (task) model, a problem domain data model, and an interaction domain model. These models are under control of a particular component, namely the virtual world control. The introduced object hierarchy supports the construction of a design base with reusable objects as well as the integration of further design knowledge, such as help concepts.

Introduction

For virtual worlds to fulfill their promise as an enrichment for man-machine dialogues it is necessary to explore well beyond the interface in the problem domain ("world") of human agents who shall "live" in or with artificial realities. The control over virtual worlds and their entities has been investigated in the fields of animation (e.g. [Feiner et al., 1990]), computation (e.g. [Vere et al., 1990]), and human-computer interaction (e.g. [Brooks, 1988, McKenna et al., 1990]). Another interestingly step twowards control has been made by modeling cognitive/emotional aspects of agents

0-7803-1363-1/93 $3.00 © 1993 IEEE

(e.g. [Bates et al., 1991]). Since there is still a huge gap between cognitive/emotional models (mostly using AI-techniques for representation and reasoning) and device-oriented virtual reality development, software engineers have the task to " 'couple' human intelligence and machine power in a single integrated system" [Woods et al., 1988].

This paper provides a way for coupling virtual world elements/mechanisms with system development. Object-oriented design is used for the conceptual unification of the previously separated conceptual entities of data modeling and virtual realities. The keys are those tasks which have to be captured by the system, as well as the interfaces where virtual agents and human users are collaborating. Task knowledge provides the required context-sensitive control about how and when to use problem domain data and 3D-interaction media. We will follow stepwise refinements according to the concepts of model-based design [Stary, 1993]. The resulting object architecture can easily be reused for further virtual world application designs.

STEP 1: Problem Domain Model

The Example. We want end users (interested persons, customers) to communicate with an virtual airline agent handling requests for flight reservation (ticketing). Human end users contact the airline agent. They issue requests on flights. A flight request is primarily composed of properties concerning the flight (the date, destination, time, etc.). The airline agent tries to find a proper flight, i.e. the request data are matched with a flight data base containing the available flights. If one or more flights match the request data, the airline agents offers them to the interested person. If a flight is accepted by the interested person, it is going to be booked, i.e. the airline agent issues a ticket for the selected flight.

We define a **problem domain** to be a structured set of data in the task domain on which tasks, such as to find and book a flight, are operating. In our example, request handling is based on the following object classes: flight, agent (artificial airline agent, interested person), and ticket. In order to map the acquired design knowledge, i.e. all structural as well as dynamic aspects to attributes and methods of object classes accurately, we have to enhance object-oriented languages with conceptual features. In addition to the tra-

ditional concepts classification, encapsulation and inheritance, we will use novel structural relationships, namely *general relationships*, *has-component relationships*, and *role-of relationships.* Relationships refer to other objects. *General relationships* refer to independent objects, *has-component relationships* to dependent objects. A *role-of relationship*, of which an object may have at most one, defines the object to represent a role of some other object.

The super class to **ProblemDomain** will be the object class **VirtualWorldControl** – the main control component of any application (see virtual world control model – step 4). The problem domain object class is related to agent tasks (which be represented by the object class **TaskDomain** in the task model – step 3), since agent tasks are based on problem domain objects. It is also related to the object class **InteractionDomain** (see interaction domain model – step 2), since problem domain data have to be presented to members of the virtual worlds for task accomplishment.

A **Flight** object class is required to match available flights with interested persons' requests, as well as to issue tickets for available flights. Besides the generic flight attributes, such as flightNumber, the object class **Flight** has a general *relationship* to the artificial airline agent, and dependent relationships to a set of interested persons and flight requests (expressed by *components*). **Agent**s have to be considered in order to specify the task accomplishment. We have to model two different kinds of agents: artificial airline agents and interested persons (end users). Usually, the human agent (end user) is able to interrupt virtual agent activities by glove actions.

In order to satisfy individual flight requests further information concerning **InterestedPerson**s (end users) is required: First, it has to be specified that an interested person is an agent. This specification is achieved by applying the *roleOf*-relationship. Then, a specific attribute is assigned to this type of person, namely interested, in order to state the status of interest concerning a particular flight. In the next step the dependency to the ticket object class is specified through applying the *components* relationship. As a consequence of this relationship the method part of the object class has to contain the operation cancelTicket, if a person is not longer interested in a certain flight. Similarly, our **ArtificialAgent** is defined as type of agent by applying the *roleOf*-relationship. If a flight is going to be booked, a ticket is issued by the airline agent. A **Ticket** object class has to be created, in order to specify the final task output.

STEP 2: The Interaction Domain Model

We consider all means which support the interaction of human users and virtual agents with elements of the virtual world to be interaction media. They set up the socalled **interaction domain**. Besides traditional 2D-interaction media panel, icon, and mouse the object subclass hierarchy for the interaction domain class has to capture 3D-interaction. For instance, the object class **3D-Device** inherits properties to its subclasses **Pointing/Picking-Device** and **InformationSpace**. A data glove is defined is assigned to the class **Pointing&PickingDevice**. Specific attributes are assigned to the glove, namely x-, y-, z-position, since the position identifies one part of the context the glove is used in. The other part of the context is provided by the position of the fingers which is evaluated by the *method* interpretFunction leading to 'pointing', 'picking up data' and transferring them, or 'dismiss information' at the current position.

InformationSpaces are 3D-'storage rooms' for information. Usually they are backed by data bases providing raw information. Information spaces provide several views and access paths to this information. For instance, flight information can be accessed via airlines, destinations and schedule periods (e.g. summer, winter). The information space is presented as a file cabinet with registers. Each register corresponds to particular view or access paths to flights. As a consequence, redundant but consistent information can be provided for task accomplishment. Another type of information spaces are boards which supports the 3D-storage of temporary results and/or inputs to the virtual worlds (see 'matchboard' and 'requestboard' in our application). Bins have the same properties as information spaces. They are used to keep intermediate information which is not longer considered to be relevant for problem solving.

The explicit specification of the agents' tasks is still missing. Tasks are the very kind of knowledge, which is required to bridge the gap between the problem domain and the interaction domain. We will introduce a concept for task-oriented interaction control at a meta-level, namely on top of the problem and interaction domain.

STEP 3: The Task Model

In the context of task-oriented applications an agent task is defined to be a problem-solving (this is its goal), agent-specific activity in a certain application domain which is directly supported by a computer application. Our task domain contains the object classes **FlightRequest** and **TicketRequest** – specified by the dependent relationship *components*.

The methods of **FlightRequest** have to comprise modification operations for flight requests, such as acquireOptions, and as a consequence of the dependency relationship, they have to include operations like addPerson. Finally, the methods recordMatch and recordMismatch are used to record (mis)matching flight requests.

Object/Behavior Diagrams (OBD) [Kappl et al., 1991] are used for the specification of the control flow. These diagrams can also be represented as objects. They are usually stored together with the static application components in a design base. An OBD represents an object class graphically by a box giving the name of the object class at its top, and its state at its bottom. The behaviour of object classes is specified by object class states and a set of class methods. Since OBDs correspond to Petri Nets, OBDs can easily be formalized and as such, implemented in a design tool.

STEP 4: Virtual World Control Model

In a virtual world where human and virtual agents collaboratively solve problems, control mechanisms have to be provided. These mechanisms concern the coordination/synchronization of interaction media handling and problem domain data manipulation, as well as passing the control from human agents to virtual agents (and vice versa) to accomplish a task. The **VirtualWorldControl** object class has the following subclasses: **ProblemDomain**, **InteractionDomain**, and **TaskDomain**.

The integration function of the control knowledge **VirtualWorldControl** is specified by the dependent relationship *components* in the object class specification. The *attribute* identifier corresponds to the name of the virtual world, and the *attribute* waiting denotes the status of the agent (application) (if currently there is no task to be executed, the application remains in a idle loop and waits for a human user input). The sequential execution of

system functions is specified by the *relationship* nextOperation. The *method* part of **VirtualWorldControl** comprises all operations which are basically required to operate a virtual world: openVirtualWorld is required in order to get into the virtual environment; its pendant exitVirtualWorld has to be executed whenever an end user intends to quit a session with the application; selectTask provides the required link to agent tasks by supporting the selection of a particular task. SelectTask indicates the complexity of control flow, since it *returns* several sets of object types: TaskDomain objects, and InteractionDomain objects. The only *preCon*dition to selectTask is to enter the virtual world.

The high level specification provided by the object class **VirtualWorld-Control** is reusable for any kind of virtual world application. It provides a clear interface to the application domain (in our strategy represented by end user tasks) which makes it generally reusable.

Conclusion

Conventional software engineering techniques do not provide proper means to integrate virtual world concepts and mechanism with traditional development concepts. The introduced approach is an attempt to make this integration step through consequent extension of a task-oriented object framework at the design level. The achieved object hierarchy improves the component-by-component development of virtual worlds for collaborative problem solving. Stepwise refinement and specific task/role allocation between human and virtual agents lead to transparency throughout the development. It also supports the explicit representation of context-sensitive control knowledge. As a consequence, a very effective way of coupling human intelligence and machine power has been achieved in an open architecture.

References

[Bates et al., 1991] Bates, J.; Loyall, A.; Reilly, S.: The Agents of Oz, in: Proceedings AAAI Spring Symposium on Integrated Intelligent Architectures, Stanford, 1991.

[Brooks, 1988] Brooks, F. Jr.: Grasping Reality Through Illusion – Interactive Graphics Serving Science, in: Proceedings ACM CHI'88, p. 1-11, 1988.

[Feiner et al., 1990] Feiner, St.; Beshers, Cl.: Worlds withing Worlds – Metaphors for Exploring n-Dimensional Virtual Worlds, in: Proceedings ACM SIGGRAPH Symposium 90, pp. 76-83, 1990.

[Kappl et al., 1991] Kappl, G.; Schrefl, M.: Object/Behaviour Diagrams, in: Proceedings IEEE 7^{th} Int. Conference on Data Engineering, pp. 530–539, Kobe, 1991.

[McKenna et al., 1990] McKenna, M.; Pieper, S.; Zeltzer, D.: Control of a Virtual Actor: The Roach, in: Proceedings 'Symposium on Interactive 3D Graphics', Snowbird, Utah, 1990.

[Stary, 1993] Stary, Ch.: Model-Based Design Bases for Task-Oriented Applications, in: Proceedings IEEE 9^{th} International Conference on Data Engineering, pp. 662-669, April 1993.

[Vere et al., 1990] Vere, S.; Bickmore, T.; A Basic Agent, in: Computational Intelligence, Vol. 6, pp. 41-60, 1990.

[Winblad et al., 1990] Winblad, A.L.; Edwards, S.D.; King, D.R.: Object-Oriented Software, Addison Wesley, Reading, Massachusetts, 1990.

[Woods et al., 1988] Woods, D.D.; Roth, J.: Cognitive Systems Engineering, in: Handbook of Human-Computer Interaction, ed.: Helander, M., pp. 3-43, North Holland, New York, 1988.